

EFFICIENT IMPLEMENTATION OF FINITE ELEMENT METHODS ON NON-MATCHING AND OVERLAPPING MESHES IN 3D

ANDRÉ MASSING , MATS G. LARSON , AND ANDERS LOGG

Abstract. In recent years, a number of finite element methods have been formulated for the solution of partial differential equations on complex geometries based on non-matching or overlapping meshes. Examples of such methods include the fictitious domain method, the extended finite element method, and Nitsche’s method. In all of these methods, integrals must be computed over cut cells or subsimplices which is challenging to implement, especially in three space dimensions. In this note, we address the main challenges of such an implementation and demonstrate good performance of a fully general code for automatic detection of mesh intersections and integration over cut cells and subsimplices. As a canonical example of an overlapping mesh method, we consider Nitsche’s method which we apply to Poisson’s equation and a linear elastic problem.

Key words. Overlapping mesh, non-matching mesh, Nitsche method, discontinuous Galerkin method, immersed interface, XFEM, algorithm, implementation, computational geometry

1. Introduction. A fundamental problem in computational science is to develop methods for the solution of partial differential equations on domains containing one or several objects that may have complex or time-dependent geometry. One approach to attacking this problem is to allow overlapping meshes where a mesh representing an object is allowed to overlap a background mesh representing the surroundings of the object; see for instance Chesshire and Henshaw [22], Mayer et al. [50], Yu [60], Zhang et al. [61] for various applications.

In solid mechanics, overlapping meshes may be used to represent materials consisting of elastic objects inserted into a surrounding elastic material of another type [37], and in fluid–structure interaction, an overlapping mesh may be used to represent an elastic body immersed in a fluid represented by a fixed background mesh [7, 49, 60]. Another common application [22, 25, 26] is found in mesh generation where a complicated geometry such as, for example, a pipe junction, may be decomposed into simpler parts and one unstructured tetrahedral mesh is created for each part. These components may then be stored, reused and recombined in applications by using an overlapping mesh technique.

Overlapping mesh techniques are of particular interest in simulations that involve moving objects. For such problems, overlapping mesh techniques are an attractive alternative to ALE techniques. The main advantage is that by using an overlapping mesh technique, one avoids deformation of the mesh that may lead to deterioration of the mesh quality and ultimately force remeshing. This is of particular importance in the simulation of fluid–structure interaction where the topology of the fluid domain may change due to deformation of the solid.

The main focus of this work is on the general algorithms and efficient implementation that is required to handle complex problems posed on overlapping meshes in three space dimensions. Many of the presented algorithms and the tools developed are of interest and use for the implementation of various overlapping mesh techniques. To make the discussion concrete, we here focus mainly on Nitsche’s method. In Hansbo et al. [33], a consistent finite element method for overlapping meshes based on Nitsche’s method was introduced and analyzed. The basic idea is to construct a finite element space by taking the direct sum of the space of continuous piecewise polynomial functions on the overlapping mesh and the restriction of the space of continuous piecewise polynomial functions to the complement of the overlapping

mesh, and then impose the interface conditions using Nitsche’s method. It was shown that this approach leads to a stable method of optimal order for arbitrary degree polynomial approximation.

The main challenge in the implementation is to compute the intersection between the overlapping and the background mesh. The result is a set of *cut* cells which may be arbitrarily complex polyhedra. These arise as the result of subtracting from the tetrahedra of the background mesh a set of tetrahedra from the overlapping mesh. By adopting algorithms and search structures from the field of computational geometry, we show how these issues can be handled in an efficient manner. Furthermore, one needs to compute integrals on the resulting polyhedra. This can be carried out efficiently based on an application of the divergence theorem in combination with potentials to represent an integral on the three-dimensional polyhedron as a sum of one-dimensional integrals on its edges.

The presented algorithms and implementation are relevant for several other types of related methods, including the extended finite element method (XFEM) [31, 58], non-fitted sharp interface methods [14, 32], and mesh-tying techniques [25].

1.1. Major contributions of this paper. Our work consists of several contributions. We identify the major techniques used in the implementation of overlapping mesh methods and related methods. We further identify useful data structures and algorithms from the field of computational geometry. As part of our work, existing computational geometry libraries such as CGAL [1] and GTS [2] have been wrapped into the general purpose finite element library DOLFIN [43] and into an extension library on top of it, thereby making these algorithms and data structures more easily accessible to the finite element community. Based on our implementation, we demonstrate for the first time a highly efficient implementation of Nitsche’s method on overlapping meshes for several problems posed in three spatial dimensions, thus opening the possibility of employing Nitsche-based overlapping mesh methods for challenging 3D problems such as fluid–structure interaction or domain-bridging problems.

1.2. Outline of this paper. In Section 2, we review Nitsche’s overlapping mesh method for a model problem and present in Section 3 the techniques and algorithms we have developed for efficient implementation of Nitsche’s method in three space dimensions. The corresponding implementation and data structures are described in Section 4. Finally, we present in Section 5 numerical examples that demonstrate the convergence of the numerical solution as well as the scaling of the work required to compute mesh intersections relative to standard finite element assembly on matching meshes.

2. Nitsche’s method on overlapping meshes. We here review Nitsche’s method for a simple model problem posed on two overlapping meshes.

2.1. Model problem. Let $\Omega_0 = (\overline{\Omega_1 \cup \Omega_2})^\circ$ be a domain in \mathbb{R}^3 , with boundary $\partial\Omega_0$, consisting of two (open and bounded) subdomains Ω_1 and Ω_2 separated by the interface $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$. We consider the following elliptic model problem: find

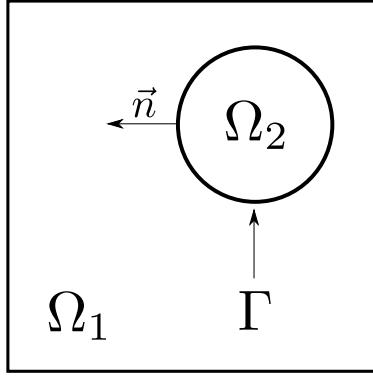


FIG. 2.1. Two domains Ω_1 and Ω_2 separated by the common interface $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$.

$u : \Omega_0 \rightarrow \mathbb{R}$ such that

$$-\Delta u_i = f_i \quad \text{in } \Omega_i, \quad i = 1, 2, \quad (2.1)$$

$$[\nabla u \cdot \mathbf{n}] = 0 \quad \text{on } \Gamma, \quad (2.2)$$

$$[u] = 0 \quad \text{on } \Gamma, \quad (2.3)$$

$$u = 0 \quad \text{on } \partial\Omega_{0,D}, \quad (2.4)$$

$$\nabla u \cdot \mathbf{n} = g \quad \text{on } \partial\Omega_{0,N}. \quad (2.5)$$

We here use the notation $v_i = v|_{\Omega_i}$ for the restriction of a function v to the subdomain Ω_i for $i = 1, 2$, \mathbf{n} is the unit normal to Γ directed from Ω_2 into Ω_1 , and $[v] = v_2 - v_1$ denotes the jump in a function over the interface Γ . In addition, the boundary $\partial\Omega_0$ is divided into two subdomains $\partial\Omega_{0,D}$ and $\partial\Omega_{0,N}$ where Dirichlet and Neumann boundary conditions are applied, respectively.

2.2. Finite element formulation. We consider a situation where a background mesh \mathcal{T}_0 is given for $\Omega_0 = (\overline{\Omega_1} \cup \overline{\Omega_2})^\circ$ and another mesh \mathcal{T}_2 is given for the overlapping domain Ω_2 (see Figure 2.1). Both meshes are assumed to consist of shape-regular tetrahedra T . The mesh \mathcal{T}_1 covering $\Omega_1 = (\Omega_0 \setminus \Omega_2)^\circ$ is constructed by

$$\mathcal{T}_1 = \{T \cap \Omega_1 : T \in \mathcal{T}_0 \wedge |T \cap \Omega_1| > 0\}. \quad (2.6)$$

Note that the \mathcal{T}_1 consists of both regular and cut elements since the mesh \mathcal{T}_0 is not a conform tetrahedralization of the subdomain Ω_1 in general. For a cut element $T \cap \Omega_1$, the degree of freedoms are defined to be the same as for the original uncut element T . This is illustrated in Figure 2.2.

We let $V_{h,i}$ denote the space of continuous piecewise fixed-order polynomials on \mathcal{T}_i that vanish on $\partial\Omega_i \cap \partial\Omega_{0,D}$ for $i = 0, 2$. The space $V_{h,1}$ is constructed as the restriction to Ω_1 of functions in $V_{h,0}$; that is, $V_{h,1} = \{v|_{\Omega_1} : v \in V_{h,0}\}$. We may then define a finite element space for the whole domain Ω_0 by

$$W_h = V_{h,1} \bigoplus V_{h,2}. \quad (2.7)$$

Nitsche' method proposed by Hansbo et al. [33] then takes the form: find $u_h \in W_h$ such that

$$a(u_h, v) = l(v) \quad \forall v \in W_h, \quad (2.8)$$

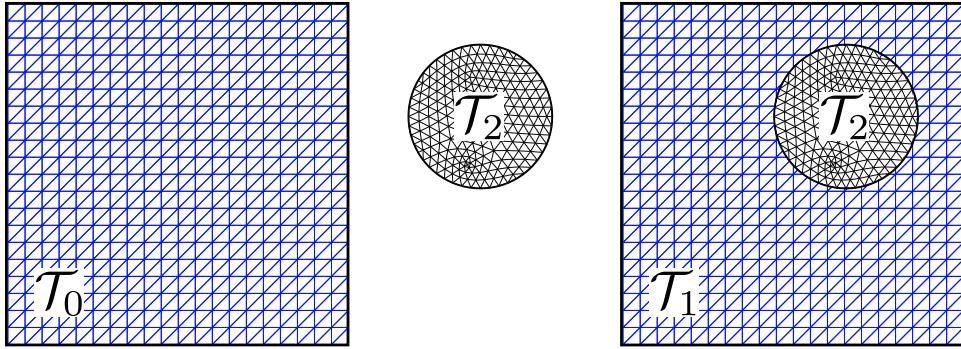


FIG. 2.2. The mesh \mathcal{T}_1 for $\Omega_1 = (\Omega_0 \setminus \Omega_2)^\circ$ is constructed as the set of cells of \mathcal{T}_0 not completely overlapped by the cells of \mathcal{T}_2 .

where

$$a(u, v) = \sum_{i=1,2} \int_{\Omega_i} \nabla u \cdot \nabla v \, dx - \underbrace{\int_{\Gamma} \langle \nabla u \cdot \mathbf{n} \rangle [v] \, dS}_{\text{Consistency}} - \underbrace{\int_{\Gamma} \langle \nabla v \cdot \mathbf{n} \rangle [u] \, dS}_{\text{Symmetrization}} + \underbrace{\gamma \int_{\Gamma} h^{-1} [u] \cdot [v] \, dS}_{\text{Penalty/Stabilization}} \quad (2.9)$$

$$l(v) = \int_{\Omega} fv \, dx + \int_{\partial\Omega_{0,N}} gv \, ds. \quad (2.10)$$

Here, γ is a positive penalty parameter and the average $\langle \nabla v \cdot \mathbf{n} \rangle$ is chosen to be the one-side derivative $\langle \nabla v \cdot \mathbf{n} \rangle = \nabla v_2 \cdot \mathbf{n}$ according to Hansbo et al. [33]. But as pointed out by the authors, any convex combination of the normal derivatives leads to a consistent formulation.

2.3. Summary of theoretical results. When the penalty parameter γ in (2.9) is chosen large enough, the form $a(\cdot, \cdot)$ is coercive on the discrete space W_h and one can derive optimal order *a priori* error estimates in both the energy norm and the L^2 norm for polynomials of arbitrary degree p . The estimates take the form

$$\left(\sum_{i=1}^2 \|\nabla(u - u_h)\|_{\Omega_i}^2 + h^{-1} \| [u_h] \|_{\Gamma}^2 \right)^{\frac{1}{2}} \leq Ch^p \|u\|_{\Omega_0, p+1}, \quad (2.11)$$

$$\|u - u_h\| \leq Ch^{p+1} \|u\|_{\Omega_0, p+1}. \quad (2.12)$$

See Hansbo et al. [33] for details. Here, $\|\cdot\|_{\Omega_0, s}$ denotes the standard Sobolev norm of order $s > 0$ on Ω_0 . *A posteriori* error estimates and adaptive algorithms are also presented in Hansbo et al. [33].

As observed by Areias and Belytschko [6], Nitsche's method for an arbitrary cutting interface as in Hansbo and Hansbo [32, 34] can be reinterpreted as a particular instance of an extended finite element method. In contrast, the presented formulation on overlapping meshes lacks such a reinterpretation since *two* unrelated meshes are involved and therefore the discontinuity across the interface cannot be modeled by the enrichment of degrees of freedom within a *single* cell. Nevertheless, both methods share some features, especially with regard to their implementation.

3. Techniques and algorithms. The main challenges in the implementation of Nitsche's method on overlapping meshes arise from the geometric computations which are necessary to assemble the discrete system associated with (2.8)–(2.10). Naturally, these geometric computations are more involved in 3D than in 2D. Similar challenges are encountered in related methods such as the extended finite element method [31], but also for the simulation of contact mechanics [59]. Different solutions have been proposed, see for example Sukumar et al. [58] in the case of extended finite elements methods, and Yang and Laursen [59] for contact problems. Here, we take another approach which efficiently solves the issues in the case of the Nitsche overlapping mesh method. In the following, we will first discuss the challenges and their remedies in general terms, and then return to the specific details of our implementation in Section 4.

3.1. Assembly on overlapping meshes. The main implementation challenges arise from the fact that the interface Γ can cross the overlapped mesh \mathcal{T}_0 in an arbitrary manner, which has two consequences. First, the definition of the finite element space (2.7) involves the restriction of the function space $V_{h,0}$, defined on the full background mesh \mathcal{T}_0 of Ω_0 , to the domain Ω_1 . The restriction results in non-standard element geometries along the interface Γ as it can be observed from the definition 2.6 of \mathcal{T}_1 . Secondly, the weak imposition of the interface conditions (2.2) and (2.3) by the interface integrals in (2.9) involves finite element spaces defined on two unrelated meshes. Both make the assembly challenging to implement, compared to a standard finite element method.

To better understand what kinds of challenges the Nitsche overlapping mesh method adds, we briefly recall the general theme of the assembly routine as it is realized nowadays in many finite element frameworks; see, e.g., Bastian et al. [12], Logg [41]. Nitsche's method is closely related to the classical discontinuous Galerkin (DG) methods, which makes it natural to depart from the assembly in the DG case. A detailed description of finite element assembly for discontinuous Galerkin methods can be found in Ølgaard et al. [55]. We assume a variational problem of the following form: find $u_h \in V_h$ such that

$$a(u_h, v) = l(v) \quad \forall v \in \hat{V}_h, \quad (3.1)$$

where a and l are bilinear and linear forms, respectively, and V_h and \hat{V}_h are the discrete trial and test spaces, respectively. For simplicity, we here make the assumption $V_h = \hat{V}_h$. The solution of the variational problem (3.1) may then be computed by solving the linear system

$$AU = b, \quad (3.2)$$

for $U \in \mathbb{R}^N$, with stiffness matrix $A_{IJ} = a(\phi_J, \phi_I)$ and load vector $b_I = l(\phi_I)$. Here, $\{\phi_I\}_{I=1}^N$ denotes a basis for V_h . During assembly of the tensors A and b , one usually iterates over all cells $T \in \mathcal{T}$ and computes the contributions from each cell as the cell tensors $A_{ij}^T = a(\phi_j^T, \phi_i^T)$ and $b_i^T = l(\phi_i)$ where $\{\phi_i^T\}_{i=1}^n$ is the local finite element basis on T (the *shape functions*). The element tensors are then scattered into the global tensors A and b by adding the entries according to a given local-to-global mapping $\iota_T : i \mapsto I$. One may similarly add contributions from each facet F of the set of boundary facets $\partial_e T$ (the set of exterior facets) and, for the implementation of a DG method, from each facet F of the set of interior facets $\partial_i T$. Algorithm 3.1 summarizes the parts of the standard assembly algorithm relevant for our discussion.

Algorithm 1 Standard finite element assembly of a bilinear form $a(\cdot, \cdot)$. The local finite element spaces are denoted by $V_h(T), V_h(F)$ etc.

```

 $A = 0$ 
for each  $T \in \mathcal{T}$ :
   $\mathcal{I}(T) = \{1, \dots, \dim(\hat{V}_h(T))\} \times \{1, \dots, \dim(V_h(T))\}$ 
  for each  $(i, j) \in \mathcal{I}(T)$ :
     $A_{ij}^T = a(\phi_j^T, \hat{\phi}_i^T)$ 
    for each  $(i, j) \in \mathcal{I}(T)$ :
       $A_{\iota_T(i), \iota_T(j)}^T + = A_{ij}^T$ 
  for each  $F \in \partial_e \mathcal{T}$ :
     $\mathcal{I}_e(F) = \{1, \dots, \dim(\hat{V}_h^e(F))\} \times \{1, \dots, \dim(V_h^e(F))\}$ 
    for each  $(i, j) \in \mathcal{I}_e(F)$ :
       $A_{ij}^T = a(\phi_j^{F,e}, \hat{\phi}_i^{F,e})$ 
    for each  $(i, j) \in \mathcal{I}_e(F)$ :
       $A_{\iota_F^e(i), \iota_F^e(j)}^T + = A_{ij}^T$ 
  for each  $F \in \partial_i \mathcal{T}$ :
     $\mathcal{I}_i(F) = \{1, \dots, \dim(\hat{V}_h^i(F))\} \times \{1, \dots, \dim(V_h^i(F))\}$ 
    for each  $(i, j) \in \mathcal{I}_i(F)$ :
       $A_{ij}^T = a(\phi_j^{F,i}, \hat{\phi}_i^{F,i})$ 
    for each  $(i, j) \in \mathcal{I}_i(F)$ :
       $A_{\iota_F^i(i), \iota_F^i(j)}^T + = A_{ij}^T$ 

```

We now consider how the standard assembly algorithm must be modified in the case of Nitsche's method on overlapping meshes. We first note that the tessellation \mathcal{T}_0 of the background domain Ω_0 may be decomposed into three disjoint subsets:

$$\mathcal{T}_0 = \mathcal{T}_{0,1} \cup \mathcal{T}_{0,2} \cup \mathcal{T}_{0,\Gamma}, \quad (3.3)$$

where $\mathcal{T}_{0,1} = \{T \in \mathcal{T}_0 : T \subset \bar{\Omega}_1\}$, $\mathcal{T}_{0,2} = \{T \in \mathcal{T}_0 : T \subset \bar{\Omega}_2\}$, and $\mathcal{T}_{0,\Gamma} = \{T \in \mathcal{T}_0 : |T \cap \Omega_i| > 0, i = 1, 2\}$ denote the sets of *not*, *completely* and *partially* overlapped cells relative to Ω_2 , respectively.

Integrals over the cells of $\mathcal{T}_{0,1}$ can be assembled using a standard assembly algorithm. Furthermore, integrals over the cells of $\mathcal{T}_{0,2}$ need not be assembled (the corresponding contributions will be assembled over \mathcal{T}_2). However, assembly must be carried out over $\mathcal{T}_{0,\Gamma}$, the partially overlapped cells of \mathcal{T}_0 . For the model problem (2.8)–(2.10), this requires the evaluation of integrals of the type

$$\int_P \nabla \phi_j^T \cdot \nabla \phi_i^T \, dx \text{ and } \int_P f \phi_i^T \, dx, \quad (3.4)$$

on *cut* elements (polyhedra) $P = T \cap \Omega_1$ where $T \in \mathcal{T}_0$. Examining (2.8)–(2.10), we further note that we must assemble the terms

$$- \int_{\Gamma} \langle \nabla \phi_j \cdot \mathbf{n} \rangle [\phi_i] \, dS - \int_{\Gamma} \langle \nabla \phi_i \cdot \mathbf{n} \rangle [\phi_j] \, dS + \gamma \int_{\Gamma} h^{-1} [\phi_j] \cdot [\phi_i] \, dS. \quad (3.5)$$

This poses an additional challenge, since the integrands involve products of trial and test functions defined on *different* meshes. Furthermore, the interface Γ consists of a subset of the boundary facets $\partial_e \mathcal{T}_2$ of \mathcal{T}_2 , but each such facet may intersect several

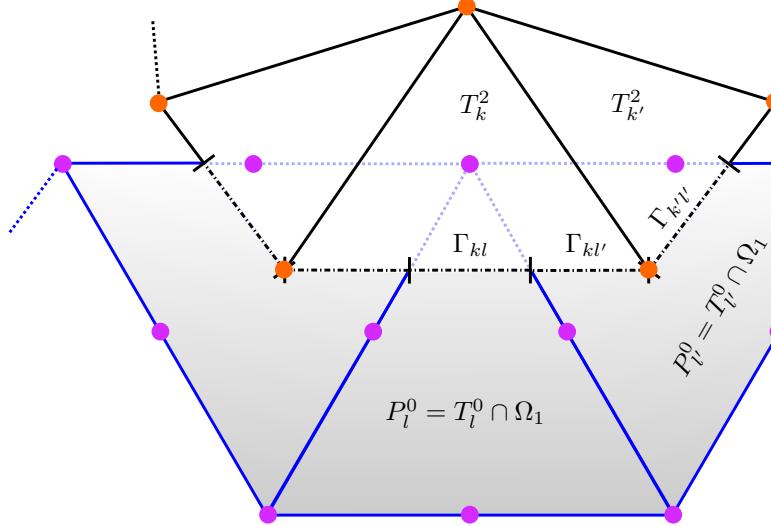


FIG. 3.1. The overlapping meshes in the intersection zone. The filled circles represent the degrees of freedom on each element (here for a piecewise linear approximation on \mathcal{T}_2 and piecewise quadratic on \mathcal{T}_0). The interface Γ is partitioned so that each part intersects exactly one cell T_k^2 of the overlapping mesh \mathcal{T}_2 and one cell T_l^0 of the background mesh \mathcal{T}_0 .

cells of \mathcal{T}_0 . We therefore partition each facet on Γ into a set of polygons $\{\Gamma_{kl}\}$ such that each polygon Γ_{kl} intersects exactly one cell T_k^2 of the overlapping mesh \mathcal{T}_2 and one cell T_l^0 of the background mesh \mathcal{T}_0 . This is illustrated in Figure 3.1. Assembly may then be carried out by summing the contributions from each polygon Γ_{kl} .

In summary, we identify the following main challenges in the implementation of Nitsche's method on overlapping meshes:

1. *collision detection*: to determine which cells are involved in the intersection between the two meshes;
2. *mesh intersection*: to compute the cut cells of the background mesh \mathcal{T}_0 represented by the polyhedra $\{P_l^0\}$ and the intersection interface represented by the polygons $\{\Gamma_{kl}\}$;
3. *integration on complex polyhedra*: to compute integrals on the polyhedra $\{P_l^0\}$ (and the polygons $\{\Gamma_{kl}\}$).

In the following, we discuss these challenges in some detail and introduce concepts, data structures and algorithms to handle them in an efficient manner. We emphasize that the proposed solutions are not limited to the implementation of Nitsche-type methods, but may also be used for the implementation of related overlapping mesh methods.

3.2. Collision detection. Topological relations between entities of a single mesh can be described by concepts of connectivity or mesh incidence as presented in Logg [42] and Bastian et al. [11]. To represent the topological relation between two overlapping (colliding) meshes, we enrich the notation of Logg [42] by the concepts of *collision relations* and *collision maps*. The collision relation $\mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2$ between \mathcal{T}_0

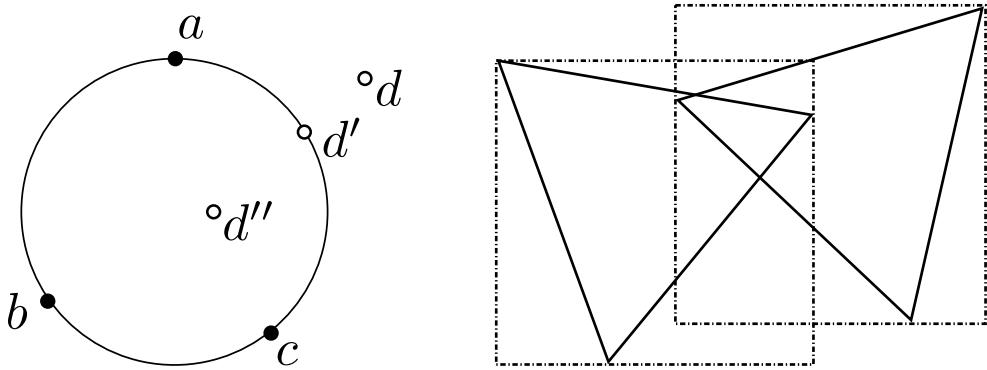


FIG. 3.2. (Left) A typical example of a geometric predicate is the incircle test, which tells whether a given point *d* lies on the inside, on the outside, or on a circle defined by three points *a*, *b*, *c*. (Right) Axis aligned bounding boxes (AABB) of two triangles.

and $\partial\mathcal{T}_2$ is defined by

$$\mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2 = \{(i, j) : T_i \cap F_j \neq \emptyset \wedge T_i \in \mathcal{T}_0 \wedge F_j \in \partial\mathcal{T}_2\}. \quad (3.6)$$

The collision relation lists pairs of indices of all intersecting cells of the background mesh \mathcal{T}_0 with boundary facets of the overlapping mesh \mathcal{T}_2 . Furthermore, each index of an intersected entity is mapped to the set of indices of intersecting entities via a pair of collision maps:

$$(\mathcal{T}_0 \rightarrow \partial\mathcal{T}_2)(i) = \{j : (i, j) \in \mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2\}, \quad (3.7)$$

$$(\partial\mathcal{T}_2 \rightarrow \mathcal{T}_0)(j) = \{i : (i, j) \in \mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2\}. \quad (3.8)$$

We note that the collision maps $\mathcal{T}_0 \rightarrow \partial\mathcal{T}_2$ and $\partial\mathcal{T}_2 \rightarrow \mathcal{T}_0$ can be computed from the collision relation $\mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2$.

A naive approach to computing the collision relation between two meshes \mathcal{T}_0 and \mathcal{T}_2 would be to intersect each cell of \mathcal{T}_0 with each cell of \mathcal{T}_2 , resulting in an $\mathcal{O}(|\mathcal{T}_0| \cdot |\mathcal{T}_2|)$ -complexity, which is not feasible for large meshes. However, efficient algorithms and data structures which reduce the complexity dramatically have been developed in the fields of computer science, computational geometry, and computer graphics. The task of determining whether two objects collide arises naturally in the rendering of a computer game and the objects in question are often represented by meshes. In this paper, we limit the description to those techniques used in our work and refer to the books Ericson [27], Schneider and Eberly [57], and Akenine-Möller et al. [3] for a broader overview. To efficiently find all pairs of the collision relation $\mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2$, two important concepts are used: (i) fast intersection tests for pairs of simple geometric objects and (ii) spatial data structures for large sets of objects to accelerate collision queries.

3.2.1. Fast intersection detection. Fast intersection detection for simple geometric entities such as triangles and tetrahedra is based on so-called *geometric predicates*. Geometric predicates are tests which determine whether two geometric entities do intersect (collide) without computing the actual intersection. See Figure 3.2 for an example. This saves work since the actual intersection does not need to be computed when the geometric predicate is false.

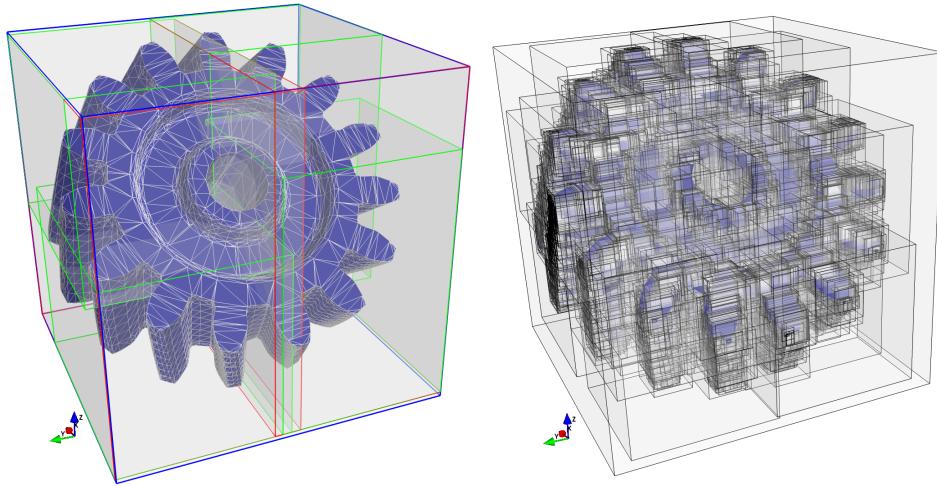


FIG. 3.3. *Bounding volume tree of gear mesh. (Left) Root bounding box (blue) and the next two generations (red and green). (Right) Every fourth level of the complete bounding box tree.*

3.2.2. Spatial data structures. Spatial data structures provide a hierarchical ordering of geometric objects, which allows quick traversal of large sets of geometric objects as part of a collision test. There exist two principal approaches based on either space subdivisions or bounding volume hierarchies, which can both be realized as tree-like data structures. As the name indicates, subdivision approaches rely on some sort of geometric subdivision of the *entire* space embedding the structures of interest, in our case a finite element mesh. Typical examples are binary space partitions (BSP) trees, quadtrees, and octrees. Since the embedding space is subdivided, the leaf of a tree will usually not represent a single mesh entity and therefore further selection procedures are required to determine the actual intersections.

In contrast, a bounding volume hierarchy is a tree which is built from bounding volumes; that is, simple geometric shapes containing the objects to be tested. Examples of such data structures are axis aligned bounding boxes (AABB, see Figures 3.2 and 3.3), oriented bounding boxes (OBB) and so-called k-DOPs, discrete orientation polytopes described by k hyperplanes. The purpose of using bounding volumes is twofold: (i) testing for collision of bounding volumes is cheaper than testing for collision of the bounded objects, and (ii) the simple geometry of the bounding volumes means that they can be stored efficiently in a hierarchical manner.

3.2.3. Building the collision map. Bounding volume trees accelerate asymmetric collision queries between a tree embedding a large set of objects and a single simple object like a tetrahedron. The concept becomes even more powerful when intersection tests between two large sets of geometric primitives are desired, as in our case of two meshes. Then a hierarchical traversal of both trees greatly improves the $\mathcal{O}(|\mathcal{T}_0| \cdot |\mathcal{T}_2|)$ -complexity of the naive approach. Algorithm 2 describes how to employ a hierarchical traversal to compute the collision map.

After the completion of the traversal according to Algorithm 2, which identifies all cells of \mathcal{T}_0 that intersect the *boundary* of \mathcal{T}_2 , it remains to check whether the remaining cells (which do not intersect the boundary of \mathcal{T}_2) are either completely

Algorithm 2 Traversal of two bounding volume trees. Starting from the root of both trees, pairs of nodes are recursively tested for intersection. Only if the bounding volumes A and B of two nodes overlap may the children possibly overlap. A so-called descend rule is applied to decide with which of the two nodes to proceed. A popular and effective rule is to choose the one with the larger volume [27] since it gives the largest volume reduction for subsequent bounding volumes. If two leaf bounding volumes are reached, then the mesh entities bounded by them are tested for intersection.

```

compute_collisions( $A, B$ ):
  if  $A \cap B = \emptyset$ :
    return
  else if is_leaf( $A$ )  $\wedge$  is_leaf( $B$ ):
    if  $T_A \cap T_B \neq \emptyset$ :
       $(i, j) = (\text{index}(T_A), \text{index}(T_B))$ 
       $\mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2 := \mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2 \cup (i, j)$ 
    else if descend_a( $A, B$ ):
      for  $a \in \text{children}(A)$ :
        compute_collisions( $a, B$ )
    else:
      for  $b \in \text{children}(B)$ :
        compute_collisions( $A, b$ )
  descend_a( $A, B$ ):
    return is_leaf( $B$ )  $\vee$  ( $\neg$ is_leaf( $A$ )  $\wedge$   $|A| \geq |B|$ )

```

overlapped or not overlapped by \mathcal{T}_2 . To check this, it is enough to take a single point contained in each cell and check whether it is in Ω_2 . To avoid building a third AABB tree for the entire mesh \mathcal{T}_2 and to take advantage of the already built tree for $\partial\mathcal{T}_2$, one can use a method known as *ray-shooting*; see Akenine-Möller et al. [3]. The idea is simple: counting how often a ray starting at the point in question intersects the surface $\partial\mathcal{T}_2$ tells whether it is inside (odd number of intersections) or outside (even number of intersections). To efficiently find all ray–surface intersections, one may reuse the same AABB tree for $\partial\mathcal{T}_2$ that was used in Algorithm 2.

To summarize, the collision map and classification of cells according to the splitting (3.3) can be found efficiently by utilizing AABB trees, fast traversal of pairs of AABB hierarchies, and ray shooting techniques.

3.3. Mesh intersection. The next step is to compute the cut cells (polyhedra) $\{P_l^0\}$ and the interface decomposition $\{\Gamma_{kl}\}$; see Figure 3.1. This computation may be phrased in terms of so-called *boolean operations* which are widely used in CAD systems to build complex geometries by performing boolean operations between primitives from a finite set of geometries. Algorithm 3 summarizes the use of boolean operations to compute $\{P_l^0\}$ and $\{\Gamma_{kl}\}$. The resulting geometric objects are depicted in Figure 3.4.

The boolean operations are completely delegated to the computational geometry library **GTS** [2] which uses a so-called *ear-clipping* algorithm to compute the surface tessellation of the intersection objects. The implemented ear clipping algorithms (first introduced in Fournier and Montuno [28]) are known to have $\mathcal{O}(n \log(n))$ complexity,

Algorithm 3 Mesh intersection. Note that the surface S defined by the union of all facets F intersecting the tetrahedron T divides T into two parts T_S^+ and T_S^- according to orientation of S .

```

for each  $T \in \mathcal{T}_0$  intersected by  $\Gamma$ :
   $l = \text{index}(T)$ 
   $P_l^0 := T$ 
   $S := \emptyset$ 
  for each  $F \in (\mathcal{T}_0 \rightarrow \partial\mathcal{T}_2)(l)$ :
     $S := S \cup F$ 
     $P_l^0 := T_S^+$ 
for each  $F \in \partial\mathcal{T}_2$ :
   $j = \text{index}(F)$ 
   $k = \text{index}(T_F)$ 
  for each  $T \in (\partial\mathcal{T}_2 \rightarrow \mathcal{T}_0)(j)$ :
     $l = \text{index}(T)$ 
     $\Gamma_{kl} := F \cap T$ 

```

where n is the number of vertices of the 3D-polygon. This number in turn (n) depends on the mesh quality parameters such as the smallest and widest element angles and will in practice be bounded by a constant.

3.4. Integration on complex polyhedra. The last required step in implementing the overlapping mesh method is to compute the integrals (3.4) and (3.5) on the cut cells $\{P_l^0\}$ and on the interface decomposition $\{\Gamma_{kl}\}$, respectively. A widely used technique in the implementation of the extended finite element method [15, 24, 52, 58] is to decompose a cell into subcells which are aligned with the interface. The interface is either approximated by a plane within the cell [29, 30] or completely recovered [24, 58], even in the case of higher order interfaces [49]. Inside each tetrahedron of the subtetrahedralization, one may then use a standard integration scheme. However, subtetrahedralization of an arbitrary polyhedron is in general a quite challenging problem [56], and its existence cannot even be guaranteed without adding additional vertices [21], in contrast to the two-dimensional case.

Methods for integration over arbitrary polygonal domains without the use of a subtriangulation have recently been presented in Natarajan et al. [53] and incorporated into the extended finite element method with discontinuous enrichments in Natarajan et al. [54]. Using Schwarz–Christoffel conformal mappings as the fundamental tool, the technique is strongly bounded to two space dimensions and hard to generalize to three space dimensions.

We here propose an alternative approach, which is based on a boundary representation of the integrals. Despite the fact that this technique has been known for a long time, see for instance [40], it seems to be largely unknown within the finite element community. Our implementation is based on the efficient realization described in Mirtich [51]. This technique may be easily generalized to the computation of the general moment integral $I_\alpha(P)$ over a polyhedron P , defined by

$$I_\alpha(P) = \int_P x^\alpha dx = \int_P x_1^{\alpha_1} \cdots x_d^{\alpha_d} dx, \quad (3.9)$$

where $\alpha = (\alpha_1, \dots, \alpha_d)$ denotes a multi-index of length d .

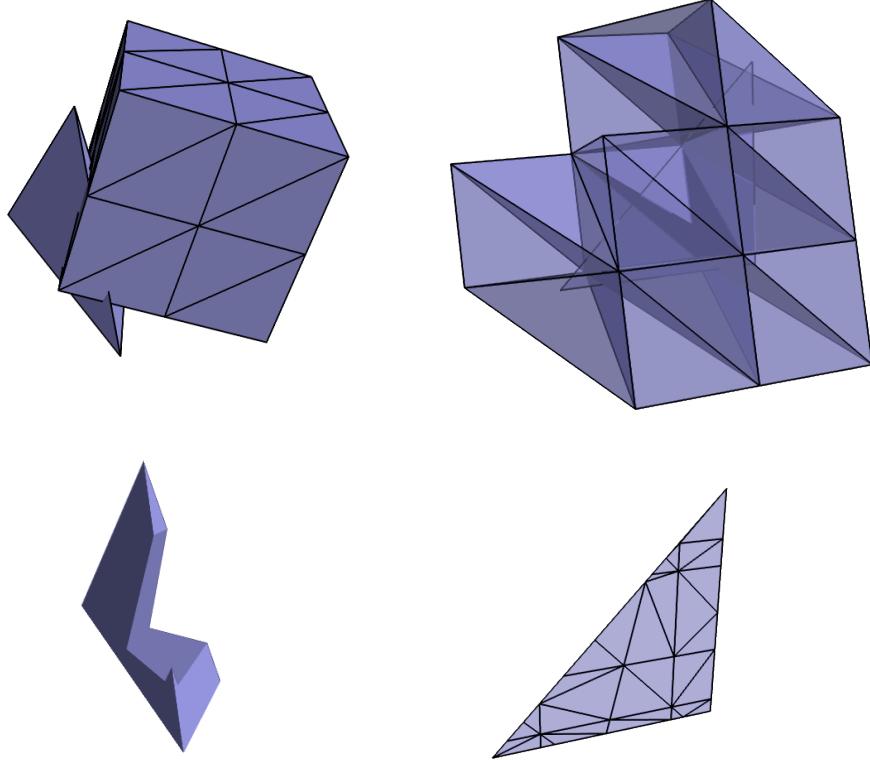


FIG. 3.4. (Left top) A tetrahedron cell T_l^0 from the background mesh T_0 overlapped by a tetrahedral mesh T_2 of a cube. (Bottom left) The resulting cut cell (polyhedron) P_l^0 . (Top right) An interface facet $F \in \partial T_2$ of the overlapping mesh T_2 intersected by a number of tetrahedra from the background mesh T_0 . (Bottom left) The resulting facet decomposition (triangulation) $\{\Gamma_{kl}\}$.

The integral $I_\alpha(P)$ is computed in three steps. The first step is to interpret the integrand as the divergence of a polynomial vector field and to rewrite (3.9) as surface integral:

$$\int_P x^\alpha \, dx = \int_P \nabla \cdot \sum_{i=1}^d \frac{x^{\alpha+\mathbf{e}_i}}{d(\alpha_i+1)} \mathbf{e}_i \, dx = \sum_{F \in \partial P} \sum_{i=1}^d \mathbf{n}_F \cdot \mathbf{e}_i \int_F \frac{x^{\alpha+\mathbf{e}_i}}{d(\alpha_i+1)} \, ds. \quad (3.10)$$

Here, $\mathbf{e}_i = (0, \dots, 0, 1, 0, \dots, 0)$ denotes the i th unit vector.

Secondly, the plane equation $ax+by+cz=d$ for each facet F allows the construction of a projection map $Z = h(X, Y)$, where X, Y, Z is some positive, orientation-preserving permutation of x, y, z . Using the parametrization h , one may rewrite the integrals of type $\int_F x^\beta \, ds$ in (3.10) as integrals in the XY -plane:

$$\int_F x^\beta \, ds = \frac{1}{|n_Z|} \int_{h^{-1}(F)} x(X, Y, h(X, Y))^\beta \, dX \, dY. \quad (3.11)$$

Here, n_Z denotes the Z -component of the normal vector \mathbf{n}_F .

The third step consists of using a parametrization of $\partial h^{-1}(F)$ and Green's theorem in the plane to rewrite (3.11) as a sum of line integrals. Together, the three

steps reduce the evaluation of the moment integral (3.9) to the evaluation of a set of one-dimensional integrals with polynomial integrands. We note that the algorithm for the calculation of each moment integral has only a $\mathcal{O}(\#F)$ complexity.

The moment integrals can be used in several ways to finally compute the integrals (3.4) and (3.5). First, one may compute the volume and the barycenter as the zeroth and first moments, respectively, which immediately provides a quadrature rule for exact integration of polynomials of degree one. Unfortunately, higher order moments may not be directly reinterpreted as quadrature rules. Furthermore, as the construction of quadrature rules for higher degree polynomials can be quite involved [23], it can be advantageous to avoid run-time generation of quadrature rules. We therefore propose to use the moment integrals $I_\alpha(P)$ directly by first interpolating the integrand onto a monomial basis and then summing the contributions:

$$\int_P f \, dx = \int_P \sum_\alpha f_\alpha x^\alpha \, dx = \sum_\alpha f_\alpha \int_P x^\alpha \, dx = \sum_\alpha f_\alpha I_\alpha(P). \quad (3.12)$$

Ill-conditioning may be avoided for higher-order expansions by replacing the monomial basis by Bernstein polynomials. In the present work, linear Lagrange elements have been used throughout and so simple barycenter quadrature suffices.

4. Implementation and data structures. We now discuss some of the data structures and classes which reflect the abstract concepts and algorithms described in Section 3. For some of these algorithms, we rely on existing implementations as part of the computational geometry libraries CGAL [1] and GTS [2], while other algorithms have been realized in the finite element library DOLFIN [43, 46] as part of this work. Specialized algorithms for the Nitsche overlapping mesh method have been implemented as part of the extension library DOLFIN-OLM which is built on top of DOLFIN. The code is free/open-source, licensed under the LGPLv3, and available at <http://launchpad.net/dolfin-olm>.

4.1. The finite element library DOLFIN. Our implementation of Nitsche's method on overlapping meshes is based on the finite element library DOLFIN which is part of the FEniCS project [41, 44] for automated scientific computing. The main feature of FEniCS is the automated treatment of finite element variational problems, based on automated generation of highly efficient C++ code from abstract high-level descriptions of finite element variational problems expressed in near-mathematical notation [4, 39]. This is combined with built-in tools for working with efficient representations of computational meshes [42] and wrappers for high-performance linear algebra libraries like PETSc [8–10] and Trilinos [36].

For this work, we have integrated the computational geometry libraries CGAL [1] and GTS [2] with DOLFIN. While CGAL and GTS provide a large part of the functionality needed to compute intersections between tetrahedra, the integration scheme was realized through an adapted version of Mirtich's code [51]. Furthermore, the assembly routine of DOLFIN was extended to handle integration over cut cells and meshes. Currently, the code for computing local interface integrals has to be implemented manually by the user, but we plan to extend FEniCS, in particular DOLFIN and the form compiler FFC [39, 45], to provide a full automation of Nitsche's method where a user only needs to supply the abstract variational problem (2.8).

In the remaining subsections, we present the class abstractions of the algorithms and data structures described in Section 3. These new classes introduced in DOLFIN-

```

void NitscheAssembler::assemble_interface(GenericTensor& A,
                                         const NitscheForm& a_nit,
                                         UFC& ufc_0,
                                         UFC& ufc_1)
{
    ...
    for (CutFacetPartIterator cut_facet(overlapping_meshes);
         !cut_facet.end(); ++cut_facet)
    {
        // Update quadrature cache to current facet part and return quadrature rule
        const QuadratureRule& quadrature =
            a_nit.interface_domain_quadrature_cache()[cut_facet->index()];
        ...

        // Get cells incident with this part of the cut cell
        std::pair<const Cell, const Cell> cells = cut_facet->adjacent_cells();
        const Cell& cell0 = cells.first;
        const Cell& cell1 = cells.second;

        // Update ufc forms and update interface local dimensions
        uint local_facet1 = cell1.index(cut_facet->entire_facet());
        ufc_0.update(cell0);
        ufc_1.update(cell1);

        // Tabulate dofs for each dimension on interface facet part
        ...

        // Tabulate interface tensor.
        a_nit.tabulate_interface_tensor(a_nit.interface_A.get(),
                                        ufc_0.cell,
                                        ufc_1.cell,
                                        local_facet1,
                                        quadrature.size(),
                                        quadrature.points(),
                                        quadrature.weights());

        // Insert matrix
        A.add(a_nit.interface_A.get(), interface_dofs);
    }
}

```

FIG. 4.1. *C++ implementation of the assembly of the so-called interface tensor accounting for the coupling between the two meshes \mathcal{T}_1 and \mathcal{T}_2 . A `CutFacetPartIterator` provides iteration over all interface facet parts, represented by `CutFacetPart` and stemming from the facet decomposition $\{\Gamma_{kl}\}$. Each `CutFacetPart` is associated with exactly one cell in each mesh, which can be accessed via the `adjacent_cells` member function. The overall design stresses the similarities to the assembly of facet contributions in the standard DG method.*

OLM allow the implementation of the Nitsche assembly algorithm in a descriptive and concise manner, as illustrated in Figure 4.1.

4.2. The class `AABBTree`. The search data structure `AABBTree` has been added to the DOLFIN library. The implementation is based on the computational geometry library CGAL [1]. Basic search queries such as finding one or all cells intersecting a given entity or distance computation are exposed via an `IntersectionOperator` class. DOLFIN-OLM complements that functionality with providing a GTS-based AABB tree [2] to allow traversal of two bounding box trees as described in Algorithm 2.

4.3. The class `OverlappingMeshes`. The class `OverlappingMeshes`, provided as part of DOLFIN-OLM, is a key component in our realization of the overlapping mesh method. It mainly computes additional topological and geometric information to describe the overlap of the two meshes \mathcal{T}_0 and \mathcal{T}_2 and provides access to this information, in particular the collision relation $\mathcal{T}_0 \leftrightarrow \partial\mathcal{T}_2$ and the collision maps $\mathcal{T}_0 \rightarrow \partial\mathcal{T}_2$ and $\partial\mathcal{T}_2 \rightarrow \partial\mathcal{T}_0$ described in Section 3.2.

A `MeshFunction`, as introduced in Logg [42], describes the splitting (3.3) of the mesh \mathcal{T}_0 by assigning different integer values to cells in the not overlapped part $\mathcal{T}_{0,1}$, the completely overlapped part $\mathcal{T}_{0,2}$, and the partially overlapped part $\mathcal{T}_{0,\Gamma}$, respectively. In the same way, the boundary facets of the overlapping mesh \mathcal{T}_2 can be marked if the overlapping domain Ω_2 is not completely contained in background domain Ω_0 .

4.4. Mesh iterators for overlapping meshes. Mesh iterators have been advocated by Berti [16, 17] and Logg [42] and used among others by Bastian et al. [12] and Botsch et al. [18] as an important abstraction concept in mesh implementations and FEM frameworks [12, 43]. The iterator concept allows to access and iterate over mesh entities such as vertices, edges, facets and cells without knowing the details of the underlying mesh implementation. We have followed the same ideas in the case of overlapping meshes. The two pairs of classes `CutCell`, `CutCellIterator` and `CutFacetPart`, `CutFacetPartIterator` provide an interface to the intersected cells in $\mathcal{T}_{0,\Gamma}$ and the interface facet partition $\{\Gamma_{kl}\}$, respectively. The `FacetPart` class and the corresponding iterator class mimic the original interface in DOLFIN by giving access to the two adjacent cells in the overlapping and the overlapped meshes, respectively. The interface assembly in Figure 4.1 presents a important use case.

Similar iterator concepts have been used in Bastian et al. [13] where a general infrastructure to couple grids interfaced by the DUNE grid framework is presented. The `CutFacetPartIterator` corresponds to the `RemoteIntersectionIterator` described in Bastian et al. [13] specialized to our case of Nitsche's method on overlapping meshes. Moreover, the `CutCellIterator` introduced here can be interpreted as an instance of `DomainIntersectionIterator` in Bastian et al. [13].

4.5. The class `Quadrature` and `QuadratureRuleCache`. The DOLFIN class `Quadrature` is a lightweight base class which only computes and stores quadrature data such as quadrature points, weights and order for a given polyhedron at runtime. Since the integration order depends on the underlying finite element scheme, the actual computation of the points and weights is meant to be implemented in subclass constructors. The class `BarycenterQuadrature` is such an instance of a subclass which computes a quadrature rule of order 2 for a given polyhedron based on the algorithm outlined in Section 3.4. For each intersected cell or facet part, a `QuadratureRule` object can be stored in a `QuadratureRuleCache` instance to save geometry and quadrature rule recomputation if several integrations have to be performed on the same intersected entity, as is the case for the computation of the stiffness matrix and load vector in the Nitsche overlapping mesh method.

4.6. Forms and assembly. The DOLFIN `Form` form class represents the mathematical concept of a finite element variational form. This class has been extended, as part of DOLFIN-OLM, to reflect the domain decomposition character of the overlapping mesh method. A so-called `NitscheForm` class holds the description of the variational problem on each part Ω_1 and Ω_2 of the domain. The coupling between the two forms is accomplished through a member function `tabulate_interface_tensor`, which computes the local interface tensor corresponding to (3.5). This is in addition

to the standard `tabulate_tensor` functions defined in the UFC code generation interface [5] for cell integrals, exterior facet integrals, and interior facet integrals (cf. Algorithm 3.1). In addition, the `NitscheForm` class gives access to the two overlapping meshes as well as a quadrature cache to avoid recomputation of quadrature rules.

Degrees of freedom of the cells of \mathcal{T}_0 that are entirely covered by the overlapping mesh \mathcal{T}_2 are *inactive*; that is, those degrees of freedom do not determine the solution and can be assigned an arbitrary value. For practical reasons, inactive degrees of freedom are included in the linear system but are set to zero by inserting 'one' on the diagonal of the corresponding rows and 'zero' in the right-hand side vector. This is automatically handled by the DOLFIN function `Matrix::ident_zeros`.

5. Numerical examples. To demonstrate the efficiency of our realization of the overlapping mesh method, we study two test examples. The first example is the Poisson equation. Here, an analytical solution for a suitable source function allows to verify the implementation by means of convergence studies. In addition, timings for the computation of both a standard P_1 finite element method and its corresponding Nitsche approximation are presented. A further breakdown of the timings in the Nitsche case gives a clear picture of the additional costs associated with the geometry-related computations and their effect on the overall computation time. From that perspective, comparing Nitsche with a simple piecewise linear, continuous finite element method represents the most challenging test case, since then the extra work required for geometry-related computations contributes the most to the overall assembly time.

As a second example, a linear elastic equation is solved with discontinuous material parameters at the interface between two overlapping meshes.

5.1. Poisson equation. We consider the elliptic model problem (2.1)–(2.5) on the domain $\Omega_0 = (0, 1)^3 \subset \mathbb{R}^3$. The overlapping domain Ω_2 is a translation and rotation of the cube $\tilde{\Omega}_2 = [0.3331, 0.6669]^3$ according to Figure 5.1. The source function f is given by

$$f(x, y, z) = 3(2\pi)^2 \sin(2\pi x) \sin(2\pi y) \sin(2\pi z). \quad (5.1)$$

Homogeneous Dirichlet boundary conditions are used on the entire boundary. The exact solution is given by

$$u(x, y, z) = \sin(2\pi x) \sin(2\pi y) \sin(2\pi z). \quad (5.2)$$

The penalty parameter γ in (2.9) is set to $\gamma = 50$. A code extract from the implementation of the solver based on DOLFIN-OLM is shown in Figure 5.1.

The Nitsche approximation was computed on a sequence of meshes with decreasing mesh size $h_{\max} = 1/N$ for the tessellations \mathcal{T}_0 and \mathcal{T}_2 , starting at $N = 14$ and stopping at $N = 104$. For the integration over cut cells and interface facets, barycenter quadrature was employed. The resulting linear systems were solved by the preconditioned conjugate gradient method as implemented in PETSc in combination with the algebraic multigrid solver from Hypre used as a preconditioner. For both the standard finite element method and the overlapping mesh method, a mesh independent number of CG iterations was observed (3–4 and 9–11, respectively). All computations were carried out on a Macbook Pro equipped with a 2.66 GHz Intel Core i7 processor

```

// Function spaces and forms on the overlapped domain
Poisson3D_1::FunctionSpace V1(mesh1);
Poisson3D_1::BilinearForm a1(V1, V1);
Poisson3D_1::LinearForm L1(V1);
L1.f = f; // assign source

// Function spaces and forms on the overlapping domain
Poisson3D_2::FunctionSpace V2(mesh2);
Poisson3D_2::BilinearForm a2(V2, V2);
Poisson3D_2::LinearForm L2(V2);
L2.f = f; // assign source

// Build Nitsche forms
PoissonNitsche::BilinearForm a_nit(a1, a2);
PoissonNitsche::LinearForm b_nit(L1, L2,
    a_nit.overlapping_meshes_ptr(),
    a_nit.overlapped_domain_quadrature_cache_ptr(),
    a_nit.interface_domain_quadrature_cache_ptr());

// Assemble
Matrix A;
Vector b;
NitscheAssembler::assemble(A, a_nit, 0, 0, 0);
NitscheAssembler::assemble(b, b_nit, 0, 0, 0);

// Apply boundary conditions
DirichletBC bc(V1, u0, boundary);
bc.apply(A, b);

// Solve linear system
Vector x;
solve(A, x, b, "cg", "amg_hypre");

// Split solution vector according to domains
Function u1(V1);
Function u2(V2);
a_nit.distribute_solution(x, u1, u2);

```

FIG. 5.1. *Code example for the Poisson problem. The domain decomposition character of Nitsche’s method is clearly reflected by defining forms on each mesh separately and “gluing” them together via a NitscheForm. Since both forms are assembled into a single matrix, the solution has to be split via the distribute_solution function.*

and 8 GB of RAM (1066 MHz DDR3). The benchmarks were repeated 10 times and averaged to obtain the reported results.

Convergence. As the theoretical results recalled in (2.11)–(2.12) predict, an optimal convergence rate in both in the H^1 - and L^2 -norm is observed; see Figure 5.4. Figure 5.3 clearly illustrates the smooth transition from the solution part u_1 defined on the overlapped mesh \mathcal{T}_1 to the part u_2 defined on the overlapping mesh \mathcal{T}_2 .

Benchmarks. The comparison in Figure 5.5 shows that the Nitsche overlapping mesh method is only twice as expensive as the standard finite element method for the same mesh size and approximately the same number of degrees of freedom. A breakdown of the computing time shows that both the assembly and the solution of the linear system become twice as expensive when the overlapping mesh method is

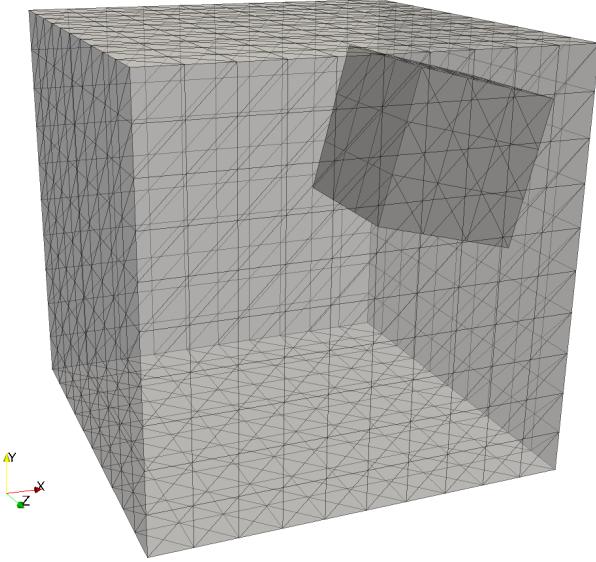


FIG. 5.2. *Mesh configuration for the Poisson problem.*

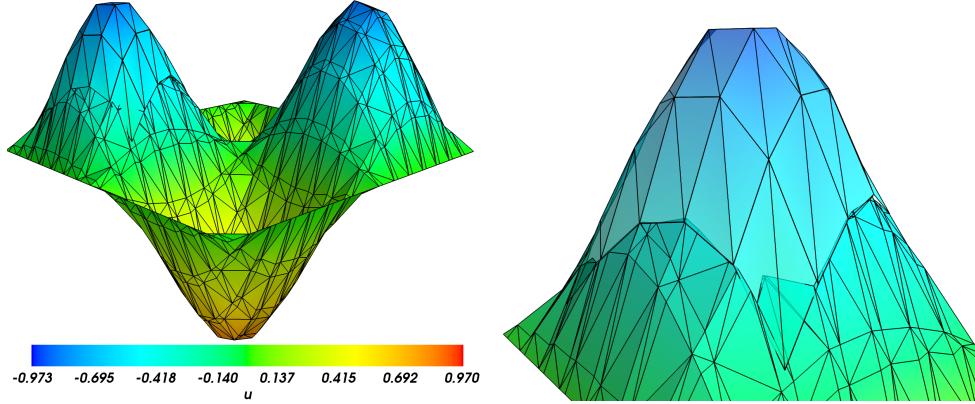


FIG. 5.3. *Two-dimensional cross-section of the three-dimensional solution of the Poisson problem showing good agreement between solutions computed on the overlapped and the overlapping mesh. (Left) Both solution parts patched together. The solution on the overlapping mesh can be seen far to the left. (Right) A zoom revealing the small discontinuity between solutions at the common interface.*

used. The latter can be attributed to the observed higher iteration numbers. The CPU time for the linear solve displays a kink in the slope at ca 3 and 4.5 million cells for the standard and Nitsche FEM, respectively, which may be attributed to the problem size increasing beyond a hardware-specific threshold (cache or memory size) that induces overhead.

A further breakdown of the assembly time is shown at the bottom of Figure 5.5. We note that while the number of cells scales like $\mathcal{O}(h^{-3})$ on a regular grid, the number of facets scales like $\mathcal{O}(h^{-2})$. Consequently, all purely interface related operations with linear complexity should scale like $N^{2/3}$, where N is the number of cells. This is also the case for the interface decomposition, computation of the cut cells, and integration

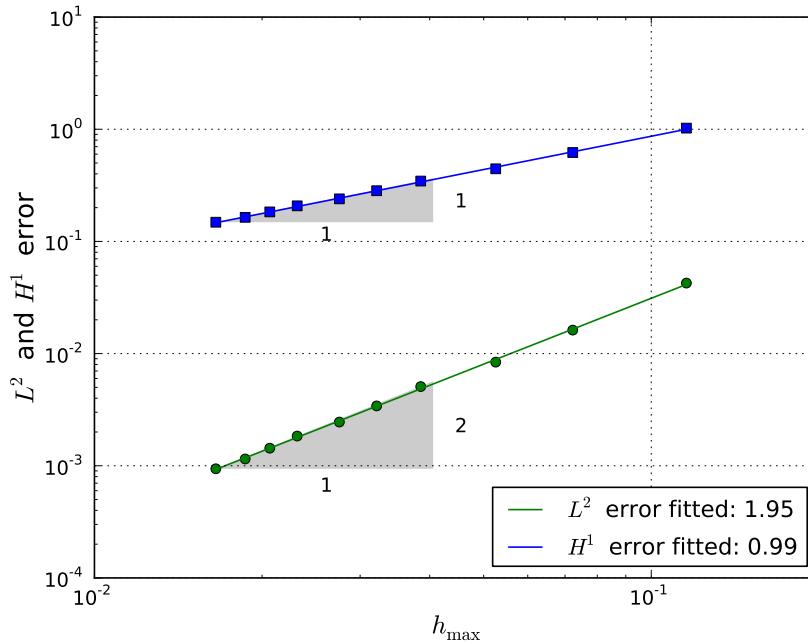


FIG. 5.4. *Convergence of the Nitsche approximation in the L^2 - and H^1 -norm for the Poisson problem.*

over the interface and the cut cells. On the other hand, the initial collision detection between the overlapped and the overlapping mesh is somewhat more expensive as it involves the searching of tree-like data structures (adding a logarithmic factor to the complexity). However, this extra cost is negligible compared to the total computing time. In conclusion, the timings depicted in Figure 5.5 indicate that the cost of assembly for the Nitsche overlapping mesh method is comparable to standard finite element assembly, and its relative efficiency increases with increasing mesh size.

A note on ill-conditioned stiffness matrices. As analyzed in Burman [19] and illustrated by numerical experiments in [47], the stiffness matrix stemming from Nitsche's method on overlapping meshes and related schemes can be ill-conditioned if the intersection gives very small elements, compared to the original element size. Different approaches have been investigated to cure the schemes from ill-conditioned systems, either by choosing proper weights for the interface [38], replacing basis functions on small elements by extensions of basis functions on larger neighboring elements, or through the use of a so-called ghost-penalty [19, 20]. For the current work, the only precaution was to skip all cut cells with a relative measure of size smaller than 10^{-15} . In our numerical experiments, we observed a mesh-independent iteration number which indicates that ill-conditioning was not present. However, proper handling of small cells by introducing ghost penalties has been studied recently for the Stokes problem in Massing et al. [48].

5.2. Linear elasticity. As a second example, we consider a linear elastic body occupying a domain Ω_0 in \mathbb{R}^3 consisting of two subdomains Ω_1 and Ω_2 with possibly different material parameters. The displacement and the stresses are assumed to

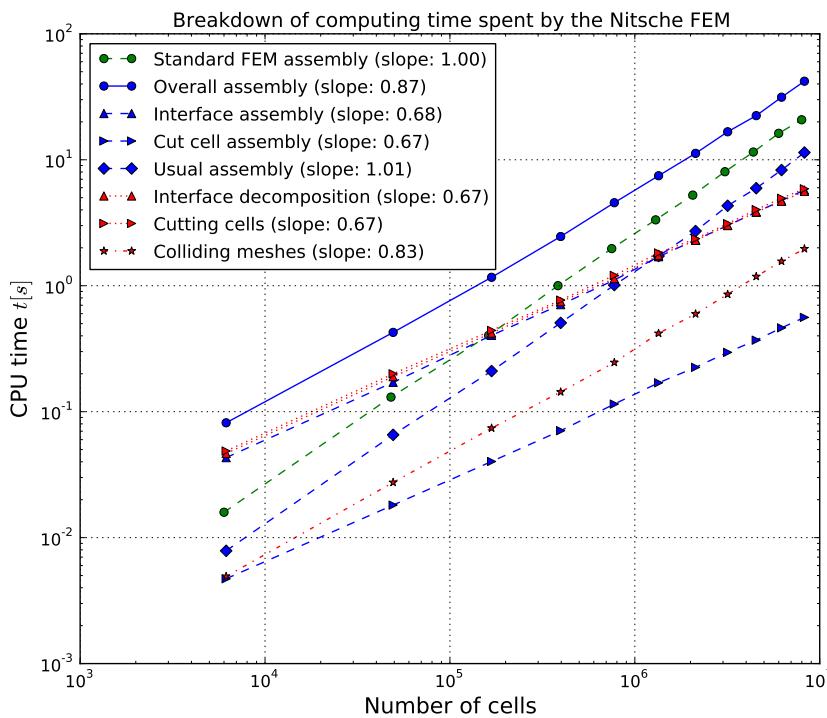
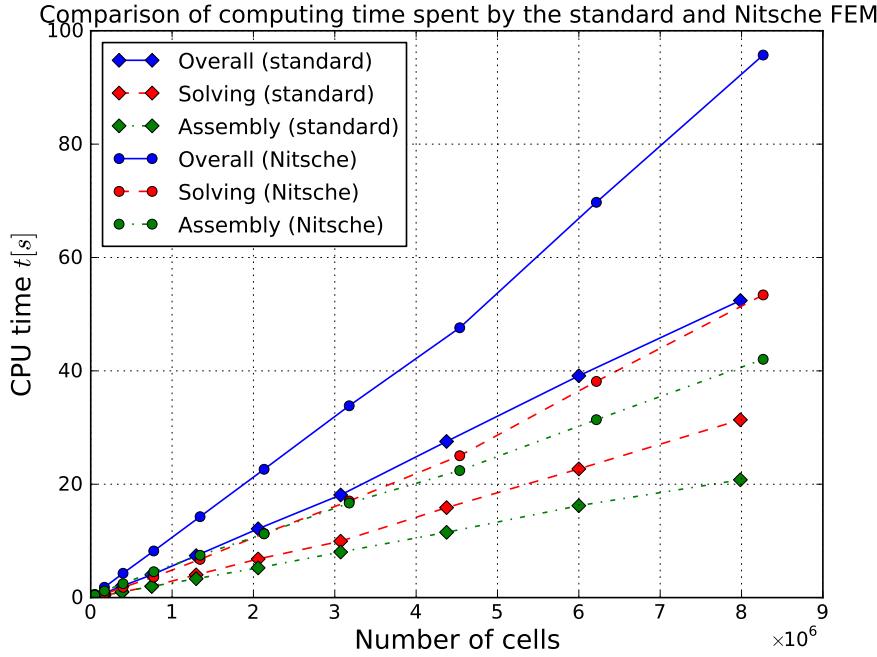


FIG. 5.5. (Top) Overall computing time spent by the standard FEM and the Nitsche FEM. (Bottom) Breakdown of the assembly time spent by the Nitsche method compared to the overall assembly time for the standard FEM. Note the expected slope of 1 for the standard (usual) assembly and of $2/3$ for the interface related operations. The graph shows that, asymptotically, overall assembly time will be dominated by standard assembly.

be continuous across the interface $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$ between the subdomains. The corresponding linear elasticity problem then takes the form: find the displacement $\mathbf{u} : \Omega_0 \rightarrow \mathbb{R}^3$ such that

$$-\operatorname{div}(\boldsymbol{\sigma}(\mathbf{u}_i)) = \mathbf{f}_i \quad \text{in } \Omega_i, \quad i = 1, 2, \quad (5.3)$$

$$[\boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n}] = \mathbf{0} \quad \text{on } \Gamma, \quad (5.4)$$

$$[\mathbf{u}] = \mathbf{0} \quad \text{on } \Gamma, \quad (5.5)$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } \partial\Omega_{0,D}, \quad (5.6)$$

$$\boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n} = \mathbf{g} \quad \text{on } \partial\Omega_{0,N}. \quad (5.7)$$

Here, the stress tensor $\boldsymbol{\sigma}$ is related to the displacement vector \mathbf{u} by Hooke's law

$$\boldsymbol{\sigma}(\mathbf{u}_i) = 2\mu_i \boldsymbol{\epsilon}(\mathbf{u}_i) + \lambda_i \operatorname{tr}(\boldsymbol{\epsilon}(\mathbf{u}_i)) \mathbf{I} \quad \text{in } \Omega_i, \quad i = 1, 2, \quad (5.8)$$

where λ_i and μ_i are the Lamé parameters in Ω_i for $i = 1, 2$ and $\boldsymbol{\epsilon}(\mathbf{v}) = (\operatorname{grad}(\mathbf{v}) + \operatorname{grad}(\mathbf{v})^\top)/2$ is the strain tensor.

Nitsche's method for the Poisson equation (2.1) proposed by Hansbo et al. [33] can be adapted to the case of the linear elastic problem (5.3)–(5.7) and takes the following form: find $\mathbf{u}_h \in \mathbf{W}_h$ such that

$$a(\mathbf{u}_h, \mathbf{v}) = l(\mathbf{v}) \quad \forall \mathbf{v} \in \mathbf{W}_h, \quad (5.9)$$

where

$$a(u, v) = \sum_{i=1}^2 \int_{\Omega_i} \boldsymbol{\sigma}(\mathbf{u}_i) : \operatorname{grad}(\mathbf{v}_i) \, dx \quad (5.10)$$

$$- \underbrace{\int_{\Gamma} \boldsymbol{\sigma}(\mathbf{u}_1) \cdot \mathbf{n} [\mathbf{v}] \, dS}_{\text{Stress balance}} - \underbrace{\int_{\Gamma} \boldsymbol{\sigma}(\mathbf{v}_1) \cdot \mathbf{n} [\mathbf{u}] \, dS}_{\text{Symmetrization}} + \underbrace{\gamma \int_{\Gamma} h^{-1} [\mathbf{u}] \cdot [\mathbf{v}] \, dS}_{\text{Penalty/Stabilization}} \quad (5.11)$$

$$l(v) = \int_{\Omega} \mathbf{f} \cdot \mathbf{v} \, dx + \int_{\partial\Omega} \mathbf{g} \cdot \mathbf{v} \, ds, \quad (5.12)$$

with γ a positive penalty parameter. Assuming that the elastic material is not nearly incompressible (λ remains bounded), we may extend the analysis in Hansbo et al. [33] and prove optimal order *a priori* error estimates. See Hansbo and Larson [35] and Becker et al. [14] for details on discontinuous Galerkin methods for elasticity problems.

Test configuration and numerical results. We consider the linear elasticity problem (5.3)–(5.7) with $\Omega_0 = (-2, 2)^3 \subset \mathbb{R}^3$ which is overlapped by a propeller-like domain $\Omega_1 = (-1, 1) \times P$ where $P = (-1, 1) \times (-0.2, 0.2) \cup (-0.2, 0.2) \times (-1, 1) \subset \mathbb{R}^2$; see Figure 3.1. The right-hand side is zero and the boundary conditions on $\partial\Omega_0$ are defined by

$$\begin{cases} \mathbf{u} = 0, & \text{on } (-2, 2)^2 \times \{-2\}, \\ \boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n} = 0, & \text{on } \partial((-2, 2)^2) \times (-2, 2), \\ \boldsymbol{\sigma}(\mathbf{u}) \cdot \mathbf{n} = \mathbf{g} & \text{on } (-2, 2)^2 \times \{2\}, \end{cases} \quad (5.13)$$

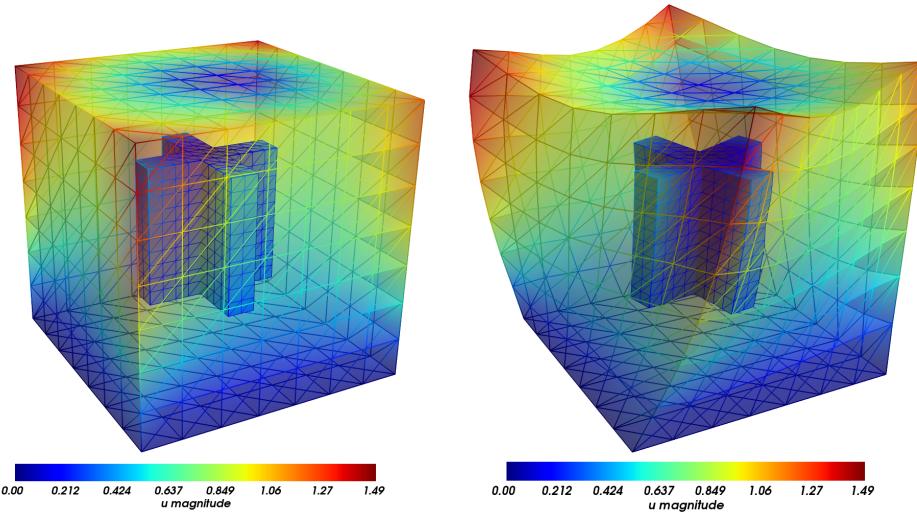


FIG. 5.6. (Left) Undeformed, original domain consisting of a cube overlaid by a propeller-like domain. (Right) The deformed domain. The color bar corresponds to the norm of the displacement \mathbf{u} .

where \mathbf{g} represents a combination of pure tangential, rotational force and a normal pressure:

$$\mathbf{g}(x, y, z) = \frac{(-y, x, 0)^\top}{5\sqrt{x^2 + y^2}} - (0, 0, 2 - \sqrt{x^2 + y^2})^\top. \quad (5.14)$$

The Lamé parameters are given by $\mu_i = E_i/(2 + 2\nu_i)$, $\lambda_i = E_i \cdot \nu_i/((1 + \nu_i)(1 - 2\nu_i))$ in Ω_i for $i = 1, 2$, with $E_1 = 10$, $E_2 = 0.1 \cdot E_1$, $\nu_1 = \nu_2 = 0.3$.

The numerical results are shown in Figures 5.6 and 5.7. The results indicate a ‘smooth’ transition of the solution from the overlapped mesh to the overlapping mesh.

6. Conclusions and outlook. We have demonstrated that overlapping mesh methods, in particular the Nitsche overlapping mesh method, may be implemented efficiently in three space dimensions through the use of tree search data structures and tools from computational geometry. Numerical tests show that optimal order convergence is obtained, that the overhead of the overlapping mesh method compared to a standard finite element method is small (roughly factor two), and that the overhead is decreasing as the size of the mesh is increased.

In the future, we plan to extend our implementation and the techniques studied in this work to handle fluid–structure interaction problems as well as contact problems. Furthermore, we plan to fully automate the implementation of Nitsche formulations on overlapping meshes by adding code generation capabilities for interface terms to FEniCS.

Acknowledgments. We thank Harish Narayanan for valuable discussions regarding the elasticity test case. This work is supported by an Outstanding Young Investigator grant from the Research Council of Norway, NFR 180450. This work is

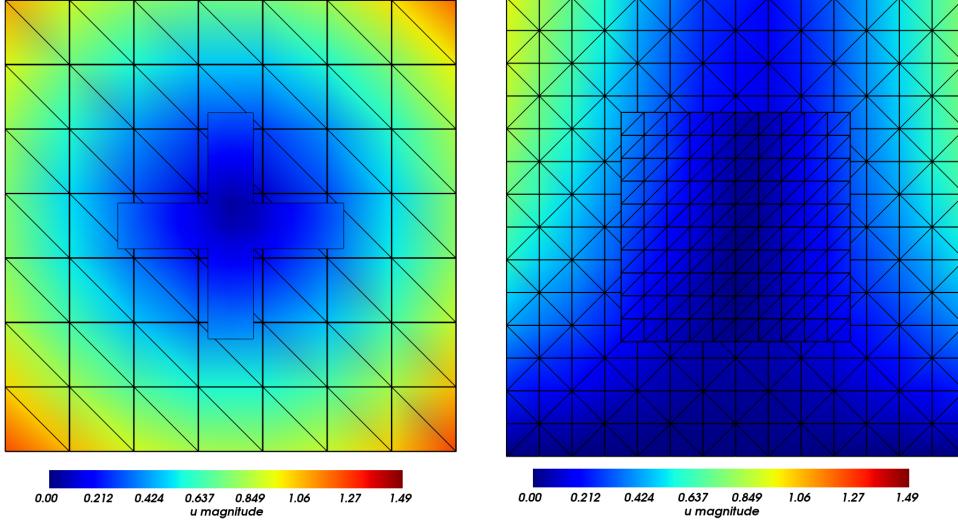


FIG. 5.7. (Left) Magnitude of the displacement \mathbf{u} in a cross-section through the xy -plane of the solution from Figure 5.6. (Right) Magnitude of the displacement \mathbf{u} in the cross-section through the xz -plane. Despite a very coarse resolution, both figures show a “smooth” transition of the solution from the domain Ω_1 to Ω_2 .

also supported by a Center of Excellence grant from the Research Council of Norway to the Center for Biomedical Computing at Simula Research Laboratory.

References.

- [1] CGAL, Computational Geometry Algorithms Library, software package. URL <http://www.cgal.org>.
- [2] GTS, GNU Triangulated Surface Library, software package. URL <http://gts.sourceforge.net/>.
- [3] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-time rendering*. A.K. Peters Ltd, 2008.
- [4] Martin S. Alnæs. *UFL: a Finite Element Form Language*, chapter 17. Springer, 2012.
- [5] Martin S. Alnæs, Anders Logg, Kent-Andre Mardal, Ola Skavhaug, and Hans Petter Langtangen. Unified framework for finite element assembly. *Int. J. Comput. Sci. Eng.*, 4(4):231–244, 2009.
- [6] P. Areias and T. Belytschko. A comment on the article “A finite element method for simulation of strong and weak discontinuities in solid mechanics” by A. Hansbo and P. Hansbo. Commentary. *Comput. Methods Appl. Mech. Engrg.*, 195(9-12):1275–1276, 2006.
- [7] J. Baiges and R. Codina. The fixed-mesh ALE approach applied to solid mechanics and fluid-structure interaction problems. *International Journal for Numerical Methods in Engineering*, 81:1529–1557, 2009.
- [8] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.
- [9] Satish Balay, Jed Brown, Kris Buschelman, Victor Eijkhout, William D. Gropp,

Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010.

- [10] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2011. URL <http://www.mcs.anl.gov/petsc>.
- [11] P. Bastian, M. Droske, C. Engwer, R. Klöfkorn, T. Neubauer, M. Ohlberger, and M. Rumpf. Towards a unified framework for scientific computing. In *Proc. of the 15th International Conference on Domain Decomposition Method*, 2004.
- [12] P. Bastian, F. Heimann, and S. Marnach. Generic implementation of finite element methods in the Distributed and Unified Numerics Environment (DUNE). *Kybernetika*, 46(2):294–315, 2010.
- [13] Peter Bastian, Gerrit Buse, and Oliver Sander. Infrastructure for the coupling of Dune grids. *Numerical Mathematics and Advanced Applications 2009*, pages 107–114, 2010.
- [14] Roland Becker, Erik Burman, and Peter Hansbo. A Nitsche extended finite element method for incompressible elasticity with discontinuous modulus of elasticity. *Comput. Methods Appl. Mech. Engrg.*, 198(41-44):3352–3360, 2009.
- [15] Ted Belytschko, Nicolas Moës, S. Usui, and C. Parimi. Arbitrary discontinuities in finite elements. *International Journal for Numerical Methods in Engineering*, 50(4):993–1013, 2001.
- [16] Guntram Berti. Generic programming for mesh algorithms: Towards universally usable geometric components. In *Proceedings of the Fifth World Congress on Computational Mechanics, Vienna University of Technology July*, 2002.
- [17] Guntram Berti. GrAL - The Grid Algorithms Library. *Future Generation Computer Systems*, 22, 2006.
- [18] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. OpenMesh - a generic and efficient polygon mesh data structure, 2002.
- [19] E. Burman. Ghost penalty. *Comptes Rendus Mathematique*, 348(21-22):1217–1220, 2010.
- [20] E. Burman and P. Hansbo. Fictitious domain finite element methods using cut elements: II. A stabilized Nitsche method. *Appl. Numer. Math.*, 62(4), 2012.
- [21] Bernard Chazelle and Leonidas Palios. Triangulating a non-convex polytope. *Discrete & Computational Geometry*, 5(1):505–526, 1990.
- [22] G. Cheshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *Journal of Computational Physics*, 90(1):1–64, 1990.
- [23] Ronald Cools. Constructing cubature formulae: the science behind the art. *Acta Numerica*, 6(-1):1–54, January 1997.
- [24] Christophe Daux, Nicolas Mos, John Dolbow, Natarajan Sukumar, and Ted Belytschko. Arbitrary branched and intersecting cracks with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 48(12):1741–1760, 2000.
- [25] D. Day and P. Bochev. Analysis and computation of a least-squares method for consistent mesh tying. *J. Comput. Appl. Math.*, 218(1):21–33, August 2008.
- [26] Hashmi Ben Dhia and Guillaume Rateau. The Arlequin method as a flexible engineering design tool. *International Journal for Numerical Methods in Engineering*, 62(11):1442–1462, 2005.
- [27] C. Ericson. *Real-time collision detection*. Morgan Kaufmann, 2005.

- [28] A. Fournier and D. Y. Montuno. Triangulating Simple Polygons and Equivalent Problems. *ACM Transactions on Graphics*, 3(2):153–174, 1984.
- [29] T. P. Fries. A corrected XFEM approximation without problems in blending elements. *International Journal for Numerical Methods in Engineering*, 75(5):503–532, 2008.
- [30] T. P. Fries. The intrinsic XFEM for two-fluid flows. *International Journal for Numerical Methods in Fluids*, 60(4):437–471, 2009.
- [31] T. P. Fries and T. Belytschko. The extended/generalized finite element method: An overview of the method and its applications. *International Journal for Numerical Methods in Engineering*, 84(April):253–304, 2010.
- [32] A. Hansbo and P. Hansbo. An unfitted finite element method, based on Nitsche’s method, for elliptic interface problems. *Comput. Methods Appl. Mech. Engrg.*, 191(47-48):5537–5552, 2002.
- [33] A. Hansbo, P. Hansbo, and Mats G. Larson. A Finite Element Method on Composite Grids based on Nitsche’s Method. *ESAIM, Math. Model. Num. Anal.*, 37(3):495–514, 2003.
- [34] Anita Hansbo and Peter Hansbo. A finite element method for the simulation of strong and weak discontinuities in solid mechanics. *Comput. Methods Appl. Mech. Engrg.*, 193(33-35):3523–3540, 2004.
- [35] P. Hansbo and M. G. Larson. Discontinuous Galerkin methods for incompressible and nearly incompressible elasticity by Nitsche’s method. *Comput. Methods Appl. Mech. Engrg.*, 191(17–18):1895–1908, 2002.
- [36] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.
- [37] M. Jirásek and T. Belytschko. Computational resolution of strong discontinuities. In *Fifth world congress on computational mechanics*, pages 7–12, 2002.
- [38] August Johansson and Mats G. Larson. A high order discontinuous Galerkin Nitsche method for elliptic problems with fictitious boundary. *submitted to Numerische Mathematik*, 2012.
- [39] Robert C. Kirby and Anders Logg. A Compiler for Variational Forms. *ACM Trans. Math. Softw.*, 32(3):417–444, 2006.
- [40] Y. T. Lee and A. A. G. Requicha. Algorithms for computing the volume and other integral properties of solids. I. known methods and open issues. *Communications of the ACM*, 25(9):635–641, 1982.
- [41] Anders Logg. Automating the finite element method. *Arch. Comput. Methods Eng.*, 14(2):93–138, 2007.
- [42] Anders Logg. Efficient representation of computational meshes. *Int. J. of Computat. Sci. Eng.*, 4(4):283–295, 2009.
- [43] Anders Logg and Garth N. Wells. DOLFIN: Automated finite element computing. *ACM Trans. Math. Softw.*, 37(2), 2010.
- [44] Anders Logg, Kent-Andre Mardal, Garth N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.
- [45] Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. *FFC: the FEniCS Form Compiler*, chapter 11. Springer, 2012.
- [46] Anders Logg, Garth N. Wells, and Johan Hake. *DOLFIN: a C++/Python finite element library*, chapter 10. Springer, 2012.

- [47] A. Massing. *Analysis and implementation of finite element methods on overlapping and fictitious domains*. PhD thesis, University of Oslo, 2012.
- [48] A. Massing, Mats G. Larson, A. Logg, and Marie E. Rognes. A stabilized Nitsche overlapping mesh method for the Stokes problem. *submitted*, 2012.
- [49] U. M. Mayer, A. Gerstenberger, and W. A. Wall. Interface handling for three-dimensional higher-order XFEM-computations in fluid-structure interaction. *International Journal for Numerical Methods in Engineering*, 79(7):846–869, 2009.
- [50] U. M. Mayer, A. Popp, A. Gerstenberger, and W. A. Wall. 3D fluid-structure-contact interaction based on a combined XFEM FSI and dual mortar contact approach. *Comput. Mech.*, 46(1):53–67, 2010.
- [51] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. *journal of graphics tools*, 1(2):31–50, 1996.
- [52] N. Moës, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *Int. J. Numer. Meth. Engrg.*, 46:131–150, 1999.
- [53] S. Natarajan, S. Bordas, and D. R. Mahapatra. Numerical integration over arbitrary polygonal domains based on Schwarz-Christoffel conformal mapping. *International Journal for Numerical Methods in Engineering*, 80(1):103–134, 2009.
- [54] Sundararajan Natarajan, D. Roy Mahapatra, and Stéphane P. A. Bordas. Integrating strong and weak discontinuities without integration subcells and example applications in an XFEM/GFEM framework. *International Journal for Numerical Methods in Engineering*, pages 269–294, July 2010.
- [55] Kristian B. Ølgaard, Anders Logg, and Garth N. Wells. Automated code generation for discontinuous galerkin methods. *SIAM Journal on Scientific Computing*, 31(2):849–864, 2008.
- [56] J. Ruppert and R. Seidel. On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra. In *Proceedings of the fifth annual symposium on Computational geometry - SCG '89*, pages 380–392, New York, New York, USA, June 1989. ACM Press.
- [57] Philip J. Schneider and David H. Eberly. *Geometric tools for computer graphics*. Morgan Kaufmann, 2003.
- [58] N. Sukumar, N. Moës, B. Moran, and T. Belytschko. Extended finite element method for three-dimensional crack modelling. *International Journal for Numerical Methods in Engineering*, 48(11):1549–1570, 2000.
- [59] B. Yang and T. A. Laursen. A contact searching algorithm including bounding volume trees applied to finite sliding mortar formulations. *Comput. Mech.*, 41(2):189–205, 2008.
- [60] Z. Yu. A DLM/FD method for fluid/flexible-body interactions. *Journal of Computational Physics*, 207(1):1–27, 2005.
- [61] L. Zhang, A. Gerstenberger, X. Wang, and W. K. Liu. Immersed finite element method. *Comput. Methods Appl. Mech. Engrg.*, 193(21-22):2051–2067, 2004.